Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

до лабораторної роботи №6

З дисципліни: «Кросплатформні засоби програмування»

На тему: «ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ»

**Варіант 5**

Виконала:

ст. гр. КІ-305

Гринь С.М.

Прийняв:

Іванов Ю.С.

Львів 2023

**Мета роботи:** оволодіти навиками параметризованого програмування мовою Java.

**Завдання:**

1. Створити параметризований клас, що реалізує предметну область задану варіантом. Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у 9 екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

**Варіант 5:** Двозв’язний список.

**Код програми:**

ClassCRM:

package KI305.Hryn.Lab6;

/\*\*

\* Клас, що представляє CRM (Customer Relationship Management) систему для обробки даних користувачів.

\*

\* @param <T> Тип даних, які обробляються CRM системою і реалізують інтерфейс Data.

\*/

class CRM<T extends Data> {

// Поле класу

/\*\*

\* Двохзв'язний список для зберігання даних користувачів в CRM системі.

\*/

private DoublyLinkedList<T> list;

/\*\*

\* Конструктор для створення нового об'єкту CRM системи з порожнім списком користувачів.

\*/

public CRM() {

list = new DoublyLinkedList<>();

}

/\*\*

\* Знайти об'єкт з максимальним віком серед користувачів CRM системи.

\*

\* @return Об'єкт з максимальним віком або null, якщо список порожній.

\*/

public T findMax() {

return list.findMax();

}

/\*\*

\* Додати дані користувача до CRM системи.

\*

\* @param data Дані користувача, які будуть додані до системи.

\*/

public void AddData(T data) {

list.add(data);

System.out.print("Element added: ");

data.print(); // Припустимо, що в Data існує метод print() для виведення інформації.

}

/\*\*

\* Видалити дані користувача за вказаним індексом з CRM системи.

\*

\* @param i Індекс користувача, якого потрібно видалити.

\*/

public void DeleteData(int i) {

list.remove(i);

}

/\*\*

\* Вивести інформацію про наступного та попереднього користувачів за вказаним індексом.

\*

\* @param index Індекс користувача, для якого потрібно вивести інформацію.

\*/

public void PrintNextAndPrevious(int index) {

list.printNextAndPrevious(index);

}

/\*\*

\* Вивести інформацію про користувача за вказаним індексом.

\*

\* @param index Індекс користувача, для якого потрібно вивести інформацію.

\*/

public void PrintUserData(int index) {

T userData = list.getUserData(index);

if (userData != null) {

System.out.println("Person Information for index " + index + ":");

userData.print(); // Припустимо, що в Data існує метод print() для виведення інформації.

} else {

System.out.println("Person not found for index " + index);

}

}

}

ClassCRMDriver:

package KI305.Hryn.Lab6;

/\*\*

\* Клас, що представляє вхідну точку програми для CRM системи.

\*/

public class CRMDriver {

/\*\*

\* Головний метод програми, який виконує демонстрацію функціоналу CRM системи.

\*

\* @param args Масив аргументів командного рядка.

\*/

public static void main(String[] args) {

// Створюємо об'єкт CRM системи

CRM<? super Data> crm = new CRM<Data>();

// Додаємо дані клієнтів та співробітників

crm.AddData(new Customer("Leanne ", "Graham ", 2, 15));

crm.AddData(new Employee("Ervin ", "Howell ", 73));

crm.AddData(new Employee("Clementine ", "Bauch ", 45));

crm.AddData(new Customer("Patricia ", "Lebsack ", 5, 27));

crm.AddData(new Customer("Chelsey ", "Dietrich ", 10, 56));

// Виводимо інформацію про користувача за вказаним індексом

crm.PrintUserData(2);

// Виводимо інформацію про попереднього та наступного користувачів за вказаним індексом

crm.PrintNextAndPrevious(2);

// Видаляємо дані за вказаним індексом

crm.DeleteData(0);

// Знаходимо користувача з максимальним віком та виводимо інформацію

Data res = crm.findMax();

System.out.print("A person with a maximum age: \n");

res.print();

}

}

Class Customer:

package KI305.Hryn.Lab6;

/\*\*

\* Клас, що представляє об'єкт клієнта в CRM системі.

\*/

class Customer implements Data {

// Поля класу

/\*\*

\* Ім'я клієнта.

\*/

private String customerName;

/\*\*

\* Прізвище клієнта.

\*/

private String customerSurname;

/\*\*

\* Кількість покупок клієнта.

\*/

private int numberOfPurchases;

/\*\*

\* Вік клієнта.

\*/

private int age;

// Конструктор

/\*\*

\* Конструктор для створення об'єкта клієнта з заданими параметрами.

\*

\* @param cName Ім'я клієнта.

\* @param cSurname Прізвище клієнта.

\* @param cNumberOfPurchases Кількість покупок клієнта.

\* @param cAge Вік клієнта.

\*/

public Customer(String cName, String cSurname, int cNumberOfPurchases, int cAge) {

customerName = cName;

customerSurname = cSurname;

numberOfPurchases = cNumberOfPurchases;

age = cAge;

}

// Методи доступу до полів

/\*\*

\* Отримати ім'я клієнта.

\*

\* @return Ім'я клієнта.

\*/

public String getCustomerName() {

return customerName;

}

/\*\*

\* Отримати прізвище клієнта.

\*

\* @return Прізвище клієнта.

\*/

public String getCustomerSurname() {

return customerSurname;

}

/\*\*

\* Встановити ім'я клієнта.

\*

\* @param name Нове ім'я клієнта.

\*/

public void setCustomerName(String name) {

customerName = name;

}

/\*\*

\* Встановити прізвище клієнта.

\*

\* @param surname Нове прізвище клієнта.

\*/

public void setCustomerSurname(String surname) {

customerSurname = surname;

}

/\*\*

\* Отримати кількість покупок клієнта.

\*

\* @return Кількість покупок клієнта.

\*/

public int getNumberOfPurchases() {

return numberOfPurchases;

}

/\*\*

\* Встановити кількість покупок клієнта.

\*

\* @param n Нова кількість покупок клієнта.

\*/

public void setNumberOfPurchases(int n) {

numberOfPurchases = n;

}

/\*\*

\* Отримати вік клієнта.

\*

\* @return Вік клієнта.

\*/

public int getAge() {

return age;

}

// Реалізація інтерфейсу Data

/\*\*

\* Порівняти клієнта за віком.

\*

\* @param c Об'єкт для порівняння з поточним клієнтом.

\* @return Результат порівняння (негативне, нуль або позитивне число).

\*/

public int compareTo(Data c) {

Integer a = age;

return a.compareTo(c.getAge());

}

/\*\*

\* Вивести інформацію про клієнта.

\*/

public void print() {

System.out.print("Customer`s Name: " + customerName + "Customer`s Surname: " + customerSurname + ", Number of purchases: " + numberOfPurchases +

", Customer`s Age: " + age + ";\n");

}

/\*\*

\* Отримати ім'я клієнта.

\*

\* @return Ім'я клієнта.

\*/

@Override

public String getName() {

return "Customer: " + customerName + customerSurname;

}

}

interface Data:

package KI305.Hryn.Lab6;

/\*\*

\* Інтерфейс, який представляє дані, що можна порівнювати за віком.

\* Класи, які реалізовують цей інтерфейс, повинні надавати інформацію про вік та надавати можливість порівнювати дані за віком.

\*/

interface Data extends Comparable<Data> {

/\*\*

\* Отримати вік об'єкта даних.

\*

\* @return Вік об'єкта даних.

\*/

public int getAge();

/\*\*

\* Вивести інформацію про об'єкт даних.

\*/

public void print();

/\*\*

\* Отримати ім'я об'єкта даних.

\*

\* @return Ім'я об'єкта даних.

\*/

public String getName();

}

Class DoublyLinkedList:

package KI305.Hryn.Lab6;

/\*\*

\* Клас, що представляє двохзв'язний список (DoublyLinkedList).

\*

\* @param <T> Тип даних, які зберігаються в списку.

\*/

class DoublyLinkedList<T extends Data> {

// Поля класу

/\*\*

\* Перший вузол (голова) списку.

\*/

private Node<T> head;

/\*\*

\* Останній вузол (хвіст) списку.

\*/

private Node<T> tail;

/\*\*

\* Конструктор для створення порожнього двохзв'язного списку.

\*/

public DoublyLinkedList() {

head = null;

tail = null;

}

/\*\*

\* Додати дані до списку.

\*

\* @param data Дані, які будуть додані до списку.

\*/

public void add(T data) {

Node<T> newNode = new Node<>(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.prev = tail;

tail.next = newNode;

tail = newNode;

}

}

/\*\*

\* Видалити елемент зі списку за вказаним індексом.

\*

\* @param index Індекс елемента, який потрібно видалити.

\*/

public void remove(int index) {

if (index < 0 || head == null) {

return;

}

if (index == 0) {

if (head == tail) {

head = null;

tail = null;

} else {

head = head.next;

head.prev = null;

}

return;

}

Node<T> current = head;

int currentIndex = 0;

while (current != null && currentIndex < index) {

current = current.next;

currentIndex++;

}

if (current != null) {

current.prev.next = current.next;

if (current.next != null) {

current.next.prev = current.prev;

} else {

tail = current.prev;

}

}

}

/\*\*

\* Знайти об'єкт з максимальним віком в списку.

\*

\* @return Об'єкт з максимальним віком або null, якщо список порожній.

\*/

public T findMax() {

if (head == null) {

return null;

}

Node<T> current = head;

T max = head.data;

while (current != null) {

if (current.data.compareTo(max) > 0) {

max = current.data;

}

current = current.next;

}

return max;

}

/\*\*

\* Вивести інформацію про наступний та попередній елементи вузла.

\*

\* @param node Вузол, для якого потрібно вивести інформацію.

\*/

public void printNextAndPrevious(Node<T> node) {

if (node == null) {

System.out.println("Node is null.");

return;

}

if (node.prev != null) {

System.out.println("Previous element: " + node.prev.data.getName());

} else {

System.out.println("No previous element (head of the list).");

}

if (node.next != null) {

System.out.println("Next element: " + node.next.data.getName());

} else {

System.out.println("No next element (tail of the list).");

}

}

/\*\*

\* Вивести інформацію про наступний та попередній елементи вузла за вказаним індексом.

\*

\* @param index Індекс вузла, для якого потрібно вивести інформацію.

\*/

public void printNextAndPrevious(int index) {

if (head == null || index < 0) {

System.out.println("Invalid index.");

return;

}

Node<T> current = head;

int currentIndex = 0;

while (current != null && currentIndex < index) {

current = current.next;

currentIndex++;

}

printNextAndPrevious(current);

}

/\*\*

\* Отримати дані користувача за вказаним індексом.

\*

\* @param index Індекс користувача, для якого потрібно отримати дані.

\* @return Об'єкт даних користувача або null, якщо індекс виходить за межі списку.

\*/

public T getUserData(int index) {

if (index < 0 || head == null) {

return null;

}

Node<T> current = head;

int currentIndex = 0;

while (current != null && currentIndex < index) {

current = current.next;

currentIndex++;

}

if (current != null) {

return current.data;

}

return null; // Повертаємо null, якщо індекс виходить за межі списку.

}

}

Class Employee:

package KI305.Hryn.Lab6;

/\*\*

\* Клас, що представляє об'єкт співробітника в CRM системі.

\*/

class Employee implements Data {

// Поля класу

/\*\*

\* Ім'я співробітника.

\*/

private String employeeName;

/\*\*

\* Прізвище співробітника.

\*/

private String employeeSurname;

/\*\*

\* Вік співробітника.

\*/

private int employeeAge;

// Конструктор

/\*\*

\* Конструктор для створення об'єкта співробітника з заданими параметрами.

\*

\* @param eName Ім'я співробітника.

\* @param eSurname Прізвище співробітника.

\* @param eAge Вік співробітника.

\*/

public Employee(String eName, String eSurname, int eAge) {

employeeName = eName;

employeeSurname = eSurname;

employeeAge = eAge;

}

// Методи доступу до полів

/\*\*

\* Отримати ім'я співробітника.

\*

\* @return Ім'я співробітника.

\*/

public String getName() {

return "Employees`s Name: " + employeeName + employeeSurname;

}

/\*\*

\* Встановити ім'я співробітника.

\*

\* @param name Нове ім'я співробітника.

\*/

public void setEmployeeName(String name) {

employeeName = name;

}

/\*\*

\* Встановити прізвище співробітника.

\*

\* @param surname Нове прізвище співробітника.

\*/

public void setEmployeeSurname(String surname) {

employeeSurname = surname;

}

/\*\*

\* Встановити вік співробітника.

\*

\* @param n Новий вік співробітника.

\*/

public void setEmployeeAge(int n) {

employeeAge = n;

}

/\*\*

\* Отримати вік співробітника.

\*

\* @return Вік співробітника.

\*/

public int getAge() {

return employeeAge;

}

// Реалізація інтерфейсу Data

/\*\*

\* Порівняти співробітника за віком.

\*

\* @param e Об'єкт для порівняння з поточним співробітником.

\* @return Результат порівняння (негативне, нуль або позитивне число).

\*/

public int compareTo(Data e) {

Integer a = employeeAge;

return a.compareTo(e.getAge());

}

/\*\*

\* Вивести інформацію про співробітника.

\*/

public void print() {

System.out.print("Employee`s Name: " + employeeName + "Employee`s Surname: " + employeeSurname + ", Employee`s Age: " + employeeAge + ";\n");

}

}

Class Node :

package KI305.Hryn.Lab6;

/\*\*

\* Клас, що представляє вузол (Node) для двохзв'язного списку (DoublyLinkedList).

\*

\* @param <T> Тип даних, які зберігаються в вузлі.

\*/

class Node<T extends Data> {

// Поля класу

/\*\*

\* Дані, які зберігаються в вузлі.

\*/

T data;

/\*\*

\* Посилання на попередній вузол.

\*/

Node<T> prev;

/\*\*

\* Посилання на наступний вузол.

\*/

Node<T> next;

/\*\*

\* Конструктор для створення об'єкту вузла з заданими даними.

\*

\* @param data Дані, які будуть збережені в вузлі.

\*/

public Node(T data) {

this.data = data;

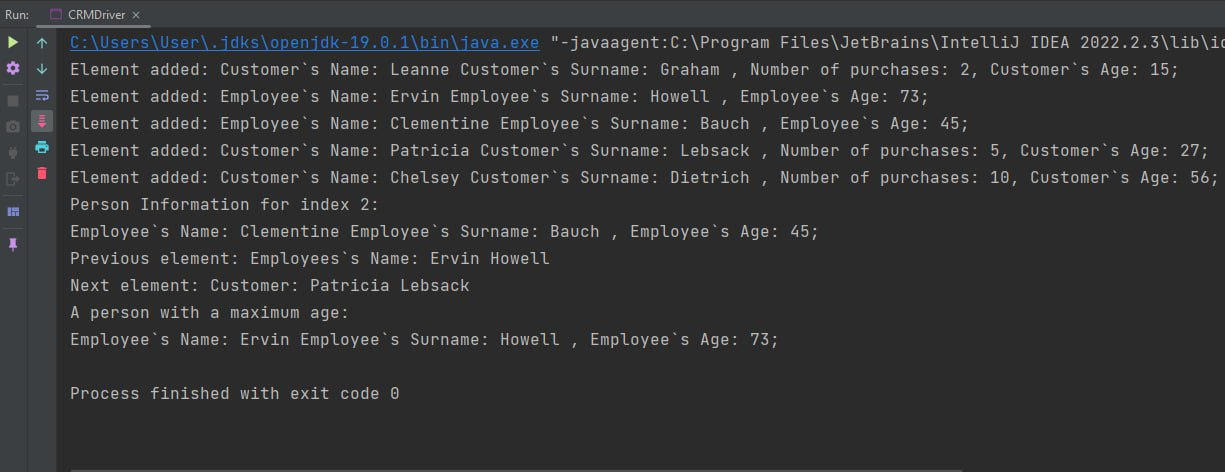
this.prev = null;

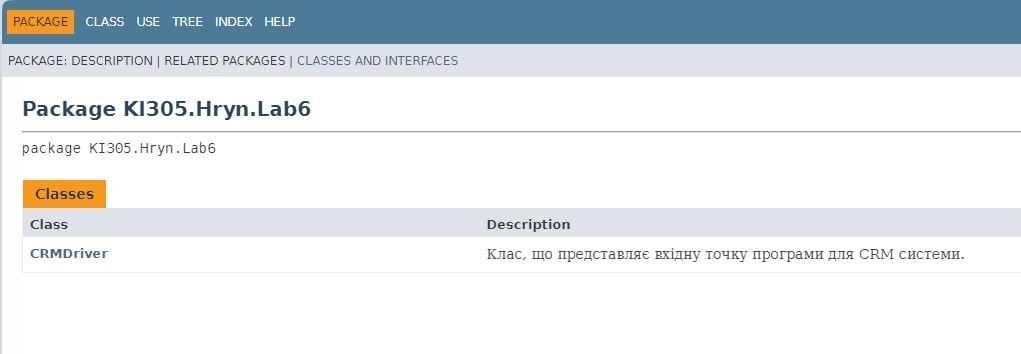
this.next = null;

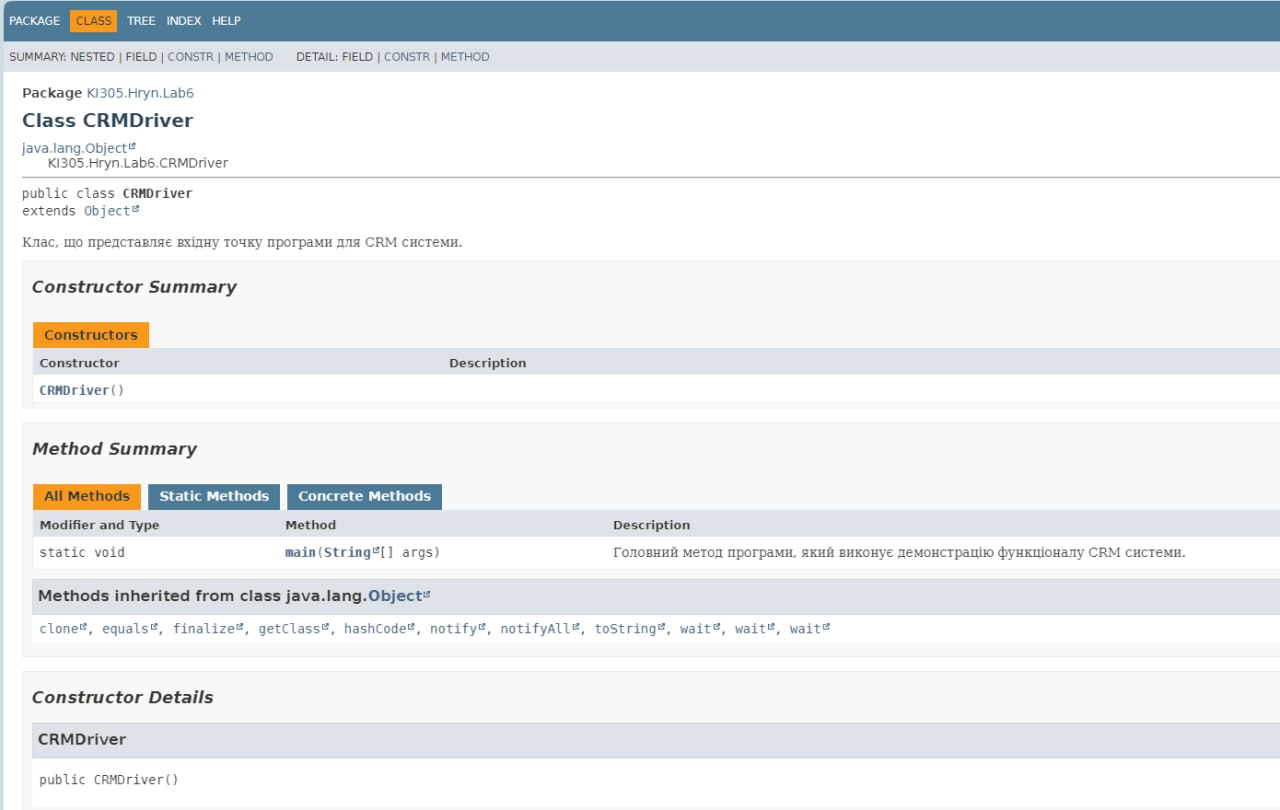
}

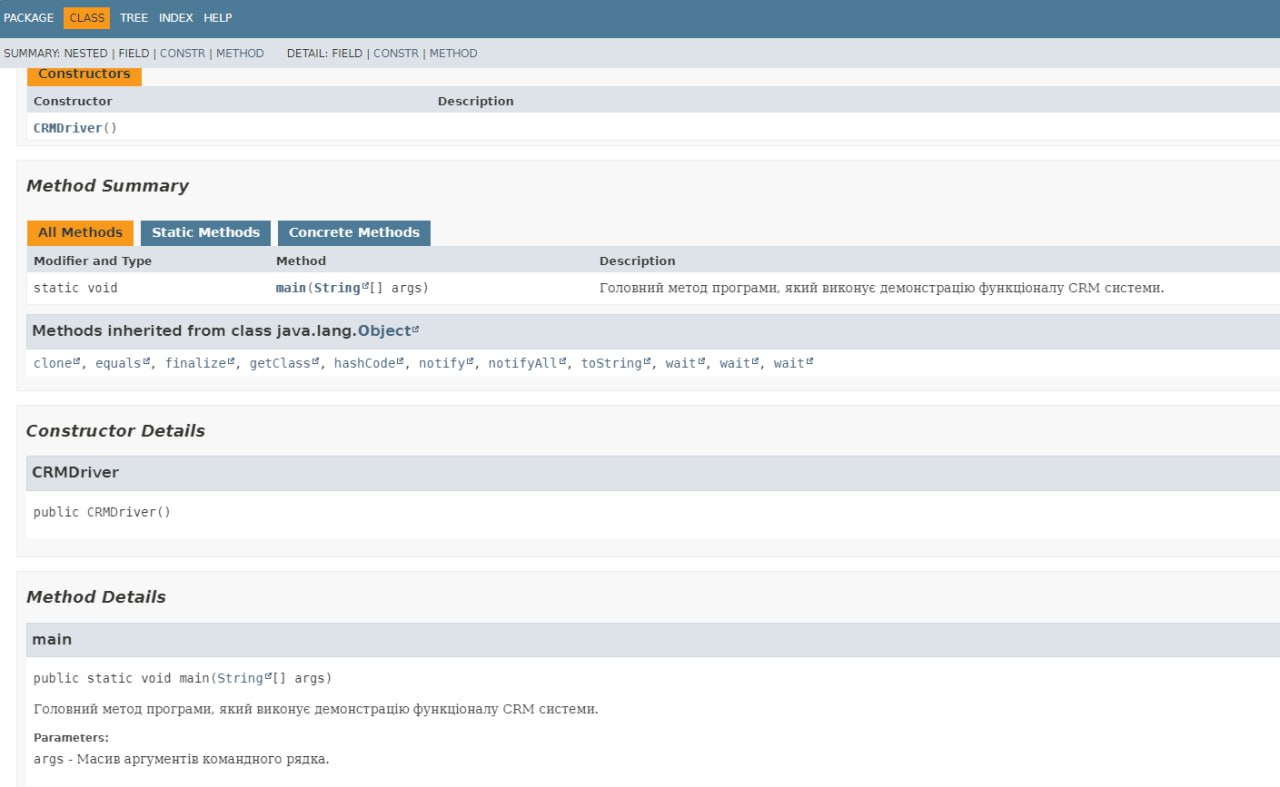
}

**Результат роботи програми:**







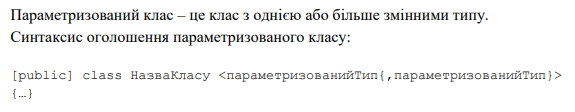


**Контрольні запитання**

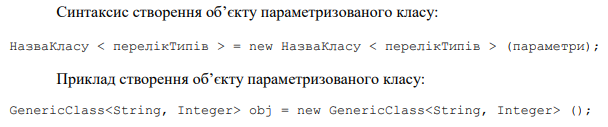
1. ***Дайте визначення терміну «параметризоване програмування».***

Параметризоване програмування є аналогом шаблонів у С++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів. Користувачів параметризованого програмування можна поділити на 3 рівні кваліфікації: 1. ті, що користуються готовими класами; 2. ті, що користуються готовими класами і вміють виправляти помилки, що виникають при їх використанні; 3. ті, що пишуть власні параметризовані класи. Для успішного застосування параметризованого програмування слід навчитися розуміти помилки, що генерує середовище при компіляції програми, що можуть стосуватися, наприклад, неоднозначності визначення спільного суперкласу для всіх переданих об’єктів. З іншої сторони необхідно передбачити захист від присвоєння об’єктів параметризованого класу, що містять об’єкти підкласу об’єктам параметризованого класу, що містять об’єкти суперкласу і дозволити зворотні присвоєння. Для вирішення цієї проблеми у мові Java введено так звані підстановочні типи. Це далеко не всі «підводні камені», що виникають при застосуванні параметризованого програмування.

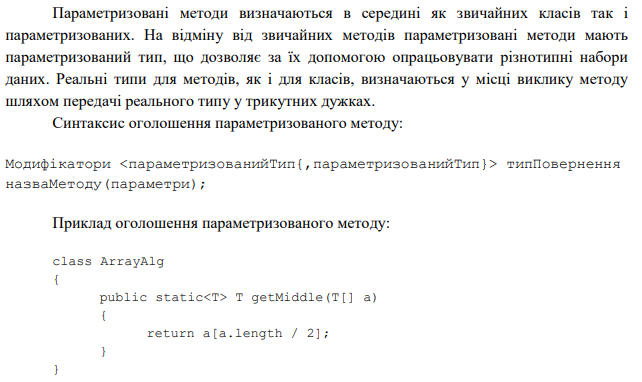
***2. Розкрийте синтаксис визначення простого параметризованого класу.***

******

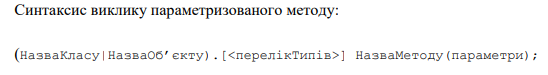
***3. Розкрийте синтаксис створення об’єкту параметризованого класу.***

******

***4. Розкрийте синтаксис визначення параметризованого методу.***

******

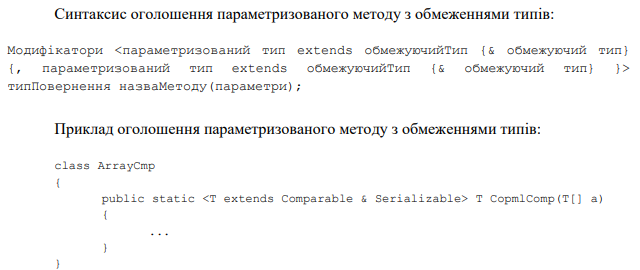
***5. Розкрийте синтаксис виклику параметризованого методу.***

******

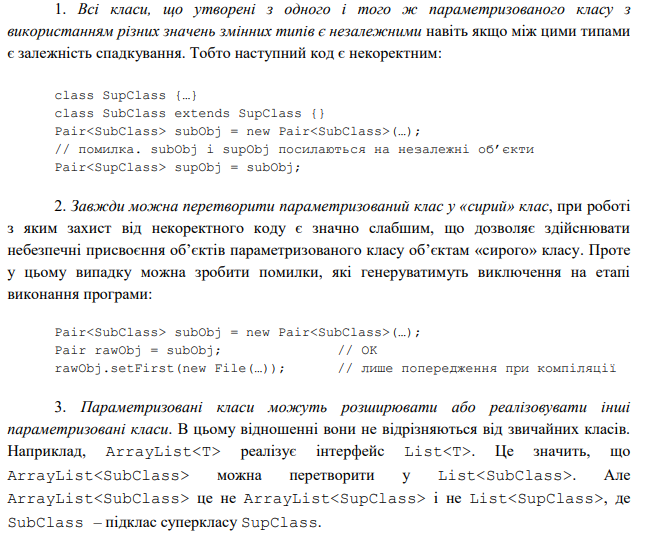
1. ***Яку роль відіграє встановлення обмежень для змінних типів?***

Бувають ситуації, коли клас або метод потребують накладення обмежень на змінні типів. Наприклад, може бути ситуація, коли метод у процесі роботи викликає з-під об’єкта параметризованого типу метод, що визначається у деякому інтерфейсі. У такому випадку немає ніякої гарантії, що цей метод буде реалізований у кожному класі, що передається через змінну типу. Щоб вирішити цю проблему у мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу. Для цього після змінної типу слід використати ключове слово extends і вказати один суперклас, або довільну кількість інтерфейсів (через знак &), від яких має походити реальний тип, що підставляється замість параметризованого типу. Якщо одночасно вказуються інтерфейси і суперклас, то суперклас має стояти першим у списку типів після ключового слова extends.

1. ***Як встановити обмеження для змінних типів?***

******

***8. Розкрийте правила спадкування параметризованих типів.***

******

***9. Яке призначення підстановочних типів?***

Підстановочні типи були введені у мову Java для збільшення гнучкості жорсткої існуючої системи параметризованих типів. На відміну від неї підстановочні типи дозволяють враховувати залежності між типами, що виступають параметрами для параметризованих типів. Це в свою чергу дозволяє застосовувати обмеження для параметрів, що підставляються замість параметризованих типів. Завдяки цьому підвищується надійність параметризованого коду, полегшується робота з ним та розділяється використання безпечних методів доступу і небезпечних модифікуючих методів. Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі main.

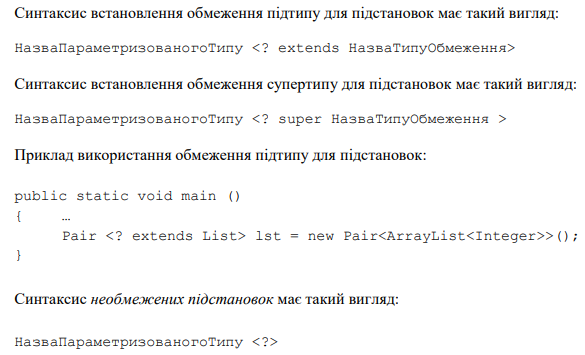
Підстановочні типи дозволяють реалізувати:

1. обмеження підтипу;

2. обмеження супертипу;

3. необмежені підстановки

***10. Застосування підстановочних типів.***

******

**Висновок:** оволоділа навиками параметризованого програмування мовою Java.